Analyse algorithmique parallèle et résultats de performance pour un solveur Navier-Stokes (psi-omega) simulant un modèle d'écoulement diphasique simplifié

Mohamed Abdelwahed, Fadi El Dabaghi, Maatoug Hassine

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Analyse algorithmique parallèle et résultats de performance pour un solveur Navier-Stokes ($\psi - \omega$) simulant un modèle d'écoulement diphasique simplifié

Mohamed Abdelwahed — Fadi Dabaghi — Maatoug Hassine

N° 4743

Février 2003

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4743--FR

INRIA
ROCQUENCOURT

Analyse algorithmique parallèle et résultats de performance pour un solveur Navier-Stokes ($\psi - \omega$) simulant un modèle d'écoulement diphasique simplifié

Mohamed Abdelwahed *, Fadi Dabaghi * , Maatoug Hassine *†

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Cosivie

Rapport de recherche n° 4743 — Février 2003 — 32 pages

Résumé : On s'intéresse dans ce travail à l'étude algorithmique parallèle et à l'implémentation sous MPI (Message Passing Interface) d'un solveur numérique simulant un écoulement diphasique eau-air modélisant l'aération mécanique dans un lac eutrophe. Ce modèle est gouverné par les équations de Navier-Stokes bidimentionnelles écrites en formulation fonction courant-tourbillon. La méthode de résolution combine la méthode des caractéristiques pour la discrétisation temporelle et une méthode d'éléments finis C^0 optimal stabilisé pour l'approximation spatiale. Le système matriciel discret obtenu est résolu par la méthode du Bigradient conjugué stabilisé avec préconditionneur. L'analyse de performance en temps de calcul du code séquentiel implémenté montre que 90% du temps est consommé par la résolution de ce système matriciel. Après une phase d'analyse algorithmique parallèle de cette partie identifiant l'interdépendance des différentes tâches et données, une implémentation du code sous MPI a été mise en place sur une machine virtuelle à mémoire distribuée. On présente la version parallélisé du code qui fût testée pour l'aspect performance. Les résultats de performance parallèle, tant sur le plan temps CPU que Elapsed, confirment l'efficacité de l'algorithme implémenté.

Mots-clés : écoulement diphasique, équations de Navier-Stokes, fonction courant-tourbillon, éléments finis, caractéristiques, calcul parallèle, mémoire distribuée, Message Passing Interface.

* INRIA Rocquencourt - Projet Cosivie
† LAMSIN, ENIT, BP37. 1002, tunis

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

A parallel algorithm analysis and performance for a $(\psi - \omega)$ Navier-Stokes solver simulating a simplified two-phase flow model

Abstract: We focus in this work on the parallel algorithmic analysis and the implemetation under MPI (Message Passing Interface) of a numerical solver simulating an air-water two-phase flow modelling an artificial mechanical aeration in eutrophized lake. This model is governed by the 2D Navier-Stokes equations written in stream function-vorticity formulation. The resolution method combines the characteristic's method for time discretization and a stabilized optimal C^o finite element method for spatial approximation. After discretization, the resulting linear system is resolved by a preconditionned Bigradient conjugate method (BICGSTAB). The numerical tests carried out on a one-processor machine showed that the resolution of the linear system by BICGSTAB is consuming about 90% of the total CPU time. Then, after an algorithmic analysis phase of this part in order to identify the independencies of the different tasks and data involved in the calculus, a parallel implementation of the solver using MPI was set up on a virtual distributed memory machine. We present a parallel version of the solver and some performance results for CPU and Elapsed time showing the efficiency of the implemented algorithm.

Key-words: Two phase flow, Navier-Stokes equations, stream function-vorticity, finite element, characteristics, parallel computing, distributed memory, Message Passing Interface.

1 Motivation et position du problème

La nécessité de préserver la qualité des eaux superficielles est aujourd'hui reconnue et devient de plus en plus pressante et cruciale. Dans le cadre de nos travaux (voir [1], [2], [8]) on s'est limité à un aspect de cette gigantesque thématique, à savoir l'étude de l'eutrophisation des lacs ou des retenues d'eau [14] et le traitement de ce fléau par aération mécanique. Sans trop s'attarder sur les causes et les effets de l'eutrophisation des retenues d'eau, sujet trop vaste pour le présent article, on peut néanmoins présenter succintement la problèmatique. Brièvement, l'eutrophisation est un processus évolutif où le facteur thermique est prépondérant; elle se manifeste par la prolifération des algues, des odeurs nauséabandes et la dégradation progressive de la qualité de l'eau. En effet dans les zones arides ou semi-arides, les variations climatiques et en particulier thermiques de l'année provoquent une division du lac en trois zones stratifiées bien distinctes :

- L'epiliminion, couche superficielle peu épaisse d'environ 7 m, brassée par les vents, où la teneur en oxygène est proche de la saturation.
- La thermocline, zone intermédiaire soumise à une chute rapide de température de 27^oC à 18^oC , de profondeur allant de 7 m à 12 m, peu soumise à l'action des vents où la teneur en oxygène est moyenne.
- L'hypolimnion, couche profonde au delà des 12 m, avec des températures de 14^oC à 18^oC, où la teneur en oxygène est faible; on y relève de fortes concentrations de gaz nocifs tels que hydrogène sulfuré, ammoniac, gaz carbonique,...etc.

Quand la concentration en oxygène devient inférieure à 3mg/l, la retenue est considérée en phase d'eutrophisation. Ceci a diverses conséquences économiques et sociales dont le fléau majeur est l'alimentation des populations en eau potable. Or, les grands lacs constituent la principale ressource des réserves d'eau potable. Face à cette situation, les techniques de restauration des lacs ou de prévention contre l'eutrophisation sont nombreuses (chimique, biologique, biochimique, mécanique,...etc): vu les coûts excessifs et les rendements relativement faibles de quelques uns de ces moyens possibles, le processus d'aération dynamique est l'un des moyens les plus efficaces et les moins onéreux. Il consiste à injecter de l'air au fond du lac (voir Figure 1) pour pousser en partie l'eau froide des couches basses vers le haut afin de l'oxygéner par contact des eaux superficielles ou par l'air atmosphérique et dans une certaine mesure par dissolution dans les couches basses; en pratique, la méthode consiste à immerger un réseau de canalisation perforées permettant à l'air comprimé envoyé dans ces tuyaux de former un rideau de bulles d'air créant la dynamique souhaitée. On se ramène donc à un problème à deux phases: l'eau et les bulles d'air.

On commence par présenter l'approche diphasique, généralement utilisée pour ce type de problème [1]. Vu la complexité de cette modélisation et la taille du problème traité, un modèle simplifié basé sur les équations de mouvement de l'eau auxquelles on ajoute l'effet des bulles d'air libérées dans l'eau sera présenté [2]. Le problème traité nécessitant toujours des capacités importantes en temps calcul et en place mémoire, on s'intéresse dans ce travail

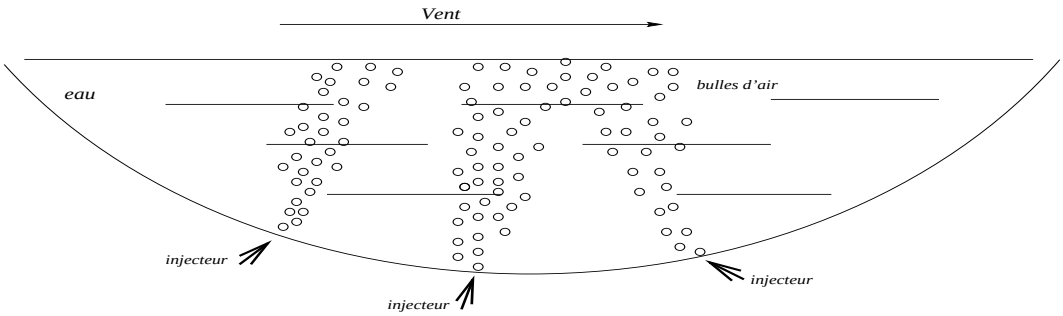

FIG. 1 – *Schéma du prossessus d'aération mécanique*

à l'étude et à la parallélisation de ce dernier modèle écrit en formulation courant-torbillon et utilisant un élément fini C^0 stabilisé . Une analyse des performances est présentée sur une machine parallèle à mémoire distribué avec 16 processeurs.

2 Aspects de modélisation

La modélisation des écoulements diphasiques pose d'énormes difficultés dûes principalement à la présence d'interface entre les phases. Mathématiquement, le domaine diphasique est constitué de zones monophasiques avec des équations de conservation vérifiants des conditions de continuité. Du point de vue physique, les processus de dispersion et de collision de phases sont compliquées à modéliser. On présente dans ce pargraphe deux approches, une approche diphasique avec ces difficultés et une approche monophasique simplifieé utilisée par la suite.

2.1 Approche diphasique

Dans cette approche, l'air et l'eau sont considérés comme deux fluides étudiés simultanément et par leur interaction ils constituent ainsi un écoulement diphasique. Le modèle le plus utilisé est le modèle à deux fluides, formulé en considérant chaque phase séparément. Ainsi, le modèle comprend des équations de conservation de la masse, de la quantité de mouvement et de l'énergie pour chaque phase, avec des termes d'interaction à l'interface qui apparaissent dans ces équations. En fait, ces équations sont obtenues en effectuant des opérations de moyenne de la formulation locale instantanée [1]; plusieurs types de moyennes sont utilisés, des moyennes temporelles (Ishii [13]), des moyennes spatiales (Enwald et al. [11]) ou des moyennes spatio-temporelles (Drew et Lahey [10]). La forme résultante de ces équations moyennés ne diffère pas sensiblement, par contre, c'est dans le sens physique des quantités moyennées et dans l'expression des termes d'échange à l'interface que résident leurs différences.
Dans ce travail, où la technique des moyennes temporelles est utilisé, les deux phases sont

considérées comme présents dans tout l'espace. Par conséquent, ces équations modélisant l'écoulement diphasique doivent être résolues simultanément dans tout le domaine représentant l'écoulement.

2.1.1 Modèle

On considère Ω un ouvert plan connexe et borné de frontière Γ simplement connexe, polygonale et lipschitzienne; on note par $Q_T = \Omega \times]0,T[$ le domaine spatio-temporel et $\Sigma_T = \Gamma \times]0,T[$ sa frontière où $T \in \mathbb{R}_+^*$.

On définit ensuite les variables suivantes :

Symboles	Variables	Type
α_k	taux de présence	scalaire
ρ_k	densité	scalaire
$\mathbf{u}_k$	vitesse	vecteur
p	pression	scalaire
T_k	température	scalaire
τ_k	contraintes de viscosité	tenseur
τ_k^t	contraintes turbulentes	tenseur
$\mathbf{j}_k$	flux de chaleur	vecteur
$\mathbf{M}_k$	moment interfacial	vecteur
$\mathbf{F}_k$	forces extérieures	vecteur

En utilisant ces notations, on présente le modèle à deux fluides basé sur un ensemble de simplifications réalistes [1]: pas d'échange de masse entre les deux phases, on néglige l'énergie cinétique turbulente ainsi que le flux de chaleur turbulent; de plus, on ne tient pas compte des contractions/expansions des bulles en supposant l'égalité de la pression sur l'interface entre les deux phases et par conséquent les bulles d'air seront sphériques à rayon constant. Dans ce cas, les équations de conservations ainsi que les lois constitutives associées s'écrivent:

- Conservation de la masse

$$\frac{\partial \alpha_k \rho_k}{\partial t} + \nabla.(\alpha_k \rho_k \, \mathbf{u}_k) = 0 \tag{1}$$

- Conservation de la quantité de mouvement

$$\frac{\partial (\alpha_k \rho_k \mathbf{u}_k)}{\partial t} + \nabla.(\alpha_k \rho_k \, \mathbf{u}_k \mathbf{u}_k) = -\alpha_k \nabla p + \nabla.(\alpha_k (\tau_k + \tau_k^t)) + \alpha_k \mathbf{F}_k + \mathbf{M}_k \tag{2}$$

- Conservation de l'énergie

$$c_{vk} \left(\frac{\partial (\alpha_k \rho_k T_k)}{\partial t} + \nabla.(\alpha_k \rho_k \, T_k \, \mathbf{u}_k) \right) = \begin{array}{l} -\nabla.(\alpha_k \, \mathbf{j}_k) + p \, \dfrac{\partial \alpha_k}{\partial t} - \alpha_k p_k \, \nabla.\mathbf{u}_k \\ +\alpha_k \tau_k : \nabla \mathbf{u}_k \end{array} \tag{3}$$

où c_{vk} est la chaleur spécifique à densité constante avec $k \in \{L, G\}, L$ désigne l'eau et G l'air.
A ces équations, on ajoutes les lois constitutives suivantes:

* Axiome de continuité

$$\alpha_L + \alpha_G = 1 \tag{4}$$

* Loi d'état

$$p = p(\rho_k, T_k) \tag{5}$$

* Terme de transfert interfacial de quantité de mouvement

$$\begin{aligned} \mathbf{M}_G = & -\mathbf{M}_L \\ = & \frac{3}{8r_B} C_D \alpha_G \rho_L |\mathbf{u}_G - \mathbf{u}_L| (\mathbf{u}_L - \mathbf{u}_G) \\ & + \alpha_G \rho_L C_{vm} \left\{ \left(\frac{\partial \mathbf{u}_L}{\partial t} + \mathbf{u}_L . \nabla \mathbf{u}_L \right) - \left(\frac{\partial \mathbf{u}_G}{\partial t} + \mathbf{u}_G . \nabla \mathbf{u}_G \right) \right\} \\ & + \alpha_G \rho_L C_l \left(\mathbf{u}_G - \mathbf{u}_L \right) \times (\nabla \times \mathbf{u}_L) \end{aligned} \tag{6}$$

où C_D est le coefficient de trainée de la bulle, r_B est le rayon moyen de bulles, C_{vm} est le coefficient de la masse ajoutée et C_l est le coefficient de portance.

* Tenseur de contraintes [13]

$$\begin{aligned} \tau_L = & \mu_L \left(\varepsilon(\mathbf{u}_L) - \frac{1}{\alpha_L} ((\nabla \alpha_L)(\mathbf{u}_G - \mathbf{u}_L)^T + (\mathbf{u}_G - \mathbf{u}_L)(\nabla \alpha_L)^T) \right) \\ \tau_G = & \mu_G \, \varepsilon(\mathbf{u}_G) \end{aligned} \tag{7}$$

où $\mu_k, (k = L, G)$ est le coefficient de viscosité et $\varepsilon(\mathbf{u}) = \nabla \mathbf{u} + (\nabla \mathbf{u})^T$

* Tenseur de contraintes turbulentes: Plusieurs modèles sont utilisés pour la modélisation du tenseur de contraintes turbulentes, parmis lesquelles, on cite le modèle à zéro équation:

$$\tau_k^t = \mu_k^t \, \varepsilon(\mathbf{u}_k)$$

où μ_k^t est la viscosité turbulente de la phase k.

* Flux de chaleur

$$\begin{aligned} \mathbf{j}_L = & -K_L \left\{ \nabla T_L - \frac{\nabla \alpha_L}{\alpha_L} (T_G - T_L) \right\} \\ \mathbf{j}_G = & -K_G \nabla T_G \end{aligned} \tag{8}$$

Enfin, on présente, le bilan des variables et des équations suivant:

Inconnues	Nombre	Equations	Nombre
α_k	2	axiome de continuité (4)	1
p	1	masse (1)	2
$\mathbf{u}_k$	6	quntité de mouvement (2)	6
ρ_k	2	loi d'état (5)	2
T_k	2	energie (3)	2
Total	13	Total	13

A ces équations s'ajoutent les conditions initiales et aux limites correspondantes.

2.1.2 Commentaires

Le modèle à deux fluides est largement utilisé dans les applications industrielles modulo des simplifications le rendant plus abordable. Les avantages de cette formulation résident dans le fait que les études et les algorithmes de résolution faits sur les équations monophasiques sont transportables aux cas diphasiques. Ceci dit, des problèmes difficiles se posent dans la modélisation du tenseur de contrainte turbulente τ_k^t ainsi que des termes de transfert interfacial écrits sur chaque interface sans oublier le coût numérique direct dû au nombre d'inconnues et la taille du problème traité.

2.2 Approche monophasique

Dans notre problème, d'énormes challenges sont posés dans l'utilisation du modèle d'écoulement présenté, autant sur le plan physique que mathématique et numérique. En effet, nous sommes face à des systèmes à grand nombre d'équations et de variables dans des cadres multi-échelles imposés par la modélisation. On rajoute un critère non négligeable au niveau numérique dû au coût énorme de calcul; ceci est dû essentiellement à la nécessité d'un maillage relativement fin du domaine étudié afin de capter un spectre significatif des échelles en jeu. Ces raisons limitent donc l'application de l'approche directe des modèles diphasiques et nous amènent à chercher une approche plus réaliste qui simule convenablement le phénomène et qui soit basé sur les modèles existants. En se basant sur le fait que l'air occupe un volume presque négligeable devant celui de l'eau, l'idée consiste représenter l'effet de la phase air sur la phase eau tout en traitant un modèle à une seule phase [1]. Il s'agit de tenir compte la dynamique des bulles: d'une part, par une condition aux limites liée à la vitesse d'injection de l'eau (ceci pour simuler le mouvement initial des bulles libérées dans l'eau) et d'autre part, par l'introduction de termes de correction diphasique. Dans ce travail on utilise l'approche simplifiée présenté par Abdelwahed et al dans [2] et Dabaghi [8] qui se limite à considérer uniquement l'effet de l'injection des bulles par une condition aux limites appliquée au niveau des injecteurs. Ce modèle dérivé de (1)-(8) s'écrit sous la forme suivante :

Trouver ρ, $\mathbf{u}$ et p solutions de

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \mathbf{u}.\nabla\rho = 0 & \text{dans } Q_T \\ \rho(.,t=0) = \rho^0(.) & \text{dans } \Omega \\ \rho\left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}.\nabla)\mathbf{u}\right) - \mu\,\Delta\mathbf{u} + \nabla p = \mathbf{F} & \text{dans } Q_T \\ \operatorname{div}\mathbf{u} = 0 & \text{dans } Q_T \\ \mathbf{u} = \mathbf{u}_d & \text{sur } \Sigma_T \\ \mathbf{u}(.,t=0) = \mathbf{u}^0(.) & \text{dans } \Omega \end{cases} \tag{9}$$

Pour l'étude numérique de ce problème on a opté d'utiliser la formulation courant-tourbillon (ψ, ω) qui permet de décrire d'une manière plus directe la dynamique du fluide à travers le calcul des lignes de courant et la vorticité. Elle a aussi l'avantage de réduire le nombre d'inconnues qui passe de trois inconnues, à savoir les deux composantes de la vitesse et la pression, à deux inconnues qui sont la fonction courant ψ et la fonction tourbillon ω. Ceci se traduit par un gain en place mémoire et en temps de calcul. Toutefois, elle se trouve confrontée à d'autres types de difficultés, entre autre, le manque de conditions aux limites sur la fonction tourbillon ω.

On introduit les fonctions courant (notée ψ) et vortivité ou tourbillon (notée ω) vérifiant :

$$\omega = \nabla \times \mathbf{u} \quad , \quad \mathbf{u} = \nabla \times \psi$$

Le système (9) s'écrit alors avec ces nouvelles variables :

Trouver ρ, ψ et ω solution de

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \mathbf{u}\nabla\rho = 0 & \text{dans } Q_T \\ \rho(.,t=0) = \rho^0(.) & \text{dans } \Omega \\ \dfrac{\partial \omega}{\partial t} + \mathbf{u}\nabla\omega - \dfrac{\mu}{\rho}\Delta\omega - \dfrac{\mu}{\rho}\left(\dfrac{\nabla\rho}{\rho}\right) \times \Delta\mathbf{u} - \dfrac{\mu}{\rho}\left(\dfrac{\nabla\rho}{\rho}\right) \times \Delta p = \nabla \times \mathbf{F} & \text{dans } Q_T \\ \omega - \nabla \times \nabla \times \psi = 0 & \text{dans } Q_T \\ \psi = \psi_d & \text{sur } \Sigma_T \\ \dfrac{\partial \psi}{\partial n} = g & \text{sur } \Sigma_T \\ \omega(.,t=0) = \omega^0(.) & \text{dans } \Omega \end{cases} \tag{10}$$

En utilisant le fait que ρ varie faiblement dans ce type d'application, on admet l'hypothèse suivante :

$$\left(\frac{\nabla\rho}{\rho}\right) \approx 0 \tag{11}$$

Utilisant cette simplification, le système (10) devient :

Trouver ρ, ψ et ω solution de

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \mathbf{u}\nabla\rho = 0 & \text{dans } Q_T \\ \dfrac{\partial \omega}{\partial t} + \mathbf{u}\nabla\omega - \dfrac{\mu}{\rho}\Delta\omega = F & \text{dans } Q_T \\ \omega + \Delta\psi = 0 & \text{dans } Q_T \\ \psi = \psi_d & \text{sur } \Sigma_T \\ \dfrac{\partial \psi}{\partial n} = g & \text{sur } \Sigma_T \\ \omega(.,t=0) = \omega^0(.) & \text{dans } \Omega \end{cases} \tag{12}$$

On s'intéresse par la suite à l'analyse et à la résolution numérique de ce système d'équations ainsi qu'à la parallélisation du solveur associé.

3 Approximation numérique

Le problème (12) est composé d'une équation de transport sur ρ et d'un système d'équations de type Navier-Stokes écrit en formulation ($\psi - \omega$). Pour la résolution numérique, on utilise la méthode des caractéristiques pour la discrétisation temporelle et la méthode des éléments finis C^0 stabilisée pour l'approximation spatiale.

3.1 Discrétisation temporelle

On va réecrire le système (12) en redéfinissant l'équation de transport et l'opérateur de convection. Pour cela, considérons le problème de Cauchy défini par

$$\begin{cases} \dfrac{\partial S}{\partial t} + \mathbf{u}\nabla S = 0 & \text{dans } Q_T \\ S(.,t=0) = S_0 & \text{dans } \Omega \end{cases} \tag{13}$$

Une discrétisation temporelle de la dérivée totale de S s'écrit au 1^{er} ordre [16] comme suit :

$$\frac{dS}{dt}(x,t^{n+1}) \simeq \frac{S(x,t^{n+1}) - S(\chi(x,t^{n+1};t^n),t^n)}{\Delta t} = \frac{S^{n+1}(x) - (S^n o \chi^n)(x)}{\Delta t} \tag{14}$$

avec χ une courbe caractéristique, solution de l'équation différentielle définie par :

$$\begin{cases} \dfrac{d}{dt}\chi(\mathbf{x},\tau;t) = \mathbf{u}(\chi(x,\tau;t),t) \\ \chi(x,t;t) = x \end{cases} \tag{15}$$

où $\chi(\mathbf{x}, \tau; t)$ est la position au temps τ de la particule qui se trouve en x au temps t.
Soit ρ^{n+1}, ψ^{n+1} et ω^{n+1} les approximations respectives de ρ, ψ et ω au temps $t = (n+1)\Delta t$.
En se servant de ce qui précède, la discrétisation en temps du problème (12) est donnée par :

Trouver ρ^{n+1}, ψ^{n+1} et ω^{n+1} solutions de

$$\begin{cases} \left(\dfrac{d\rho}{dt}\right)^{n+1} = 0 & \text{dans } \Omega \\ \left(\dfrac{d\omega}{dt}\right)^{n+1} - \dfrac{\mu}{\rho^{n+1}}\Delta\omega^{n+1} = F^{n+1} & \text{dans } \Omega \\ \omega^{n+1} + \Delta\psi^{n+1} = 0 & \text{dans } \Omega \\ \psi^{n+1} = \psi_d & \text{sur } \Gamma \\ \dfrac{\partial\psi^{n+1}}{\partial n} = g & \text{sur } \Gamma \end{cases} \tag{16}$$

Dans la formulation (16), l'équation de transport sur ρ et le terme de convection sur ω seront traités comme une dérivée totale. Ainsi la non-linéarité dans les équations de Navier-Stokes se trouve reporté dans une dérivée particulaire.
En utilisant (14) dans la première équation de (16) et connaissant ρ^n, on obtient :

$$\rho^{n+1} = \rho^n o \chi^n \tag{17}$$

avec $\chi^n = \chi(x, t^{n+1}; t^n)$.
De même pour le terme de convection sur ω, connaissant ω^n, on se ramène à la résolution du système suivant :

Trouver ψ^{n+1} et ω^{n+1} solutions de

$$\begin{cases} \dfrac{\omega^{n+1}}{\Delta t} - \dfrac{\mu}{\rho^{n+1}}\Delta\omega^{n+1} = F^{n+1} + \dfrac{\omega^n o \chi^n}{\Delta t} & \text{dans } \Omega \\ \omega^{n+1} + \Delta\psi^{n+1} = 0 & \text{dans } \Omega \\ \psi^{n+1} = \psi_d & \text{sur } \Gamma \\ \dfrac{\partial\psi^{n+1}}{\partial n} = g & \text{sur } \Gamma \end{cases} \tag{18}$$

En remplaçant le terme $\dfrac{\omega^{n+1}}{\Delta t}$ par $-\dfrac{\Delta\psi^{n+1}}{\Delta t}$ dans la première équation de (18), on obtient le problème dual suivant :

$$\begin{cases} \omega^{n+1} + \Delta\psi^{n+1} = 0 & \text{dans } \Omega \\ \Delta\omega^{n+1} + \dfrac{\rho^{n+1}}{\mu\Delta t}\,\Delta\psi^{n+1} = f^{n+1} & \text{dans } \Omega \\ \psi^{n+1} = \psi_d & \text{sur } \Gamma \\ \dfrac{\partial\psi^{n+1}}{\partial n} = 0 & \text{sur } \Gamma \end{cases} \tag{19}$$

où

$$f^{n+1} = -\frac{\rho^{n+1}}{\mu}\left(F^{n+1} + \frac{\omega^n o\chi^n}{\Delta t}\right) \tag{20}$$

A chaque pas de temps, le problème (19) est du type Quasi-Stokes donné par :

$$\begin{cases} \omega + \Delta\psi = 0 & \text{dans} \quad \Omega \\ \Delta\omega + \lambda\Delta\psi = f & \text{dans} \quad \Omega \\ \psi = \psi_d & \text{sur} \quad \Gamma \\ \dfrac{\partial\psi}{\partial n} = g & \text{sur} \quad \Gamma \end{cases} \tag{21}$$

où ψ_d, g et f sont donnés; λ est considérée par la suite égale à une constante positive.

Le problème (21) admet une solution unique (Amara et *al.* [6]).

3.2 Approximation spatiale

3.2.1 Discrétisation

Soit $\mathcal{T}_h$ une famille de triangulations régulières de $\overline{\Omega}$. On note, pour chaque triangle K de $\mathcal{T}_h$, h_K son diamètre et $mesK$ son aire et on suppose qu'il existe une constante $c > 0$ telle que :

$$\forall h_K > 0, \quad \forall K \in \mathcal{T}_h \quad h_K^2 \leq c\, mesK. \tag{22}$$

On note aussi

$$h = \max_{K\in\mathcal{T}_h} h_K \quad \text{et} \quad h_{min} = \min_{K\in\mathcal{T}_h} h_K. \tag{23}$$

A la décomposition $\mathcal{T}_h$, on associe l'ensemble $\mathcal{C}_h$ des côtés internes de $\mathcal{T}_h$. On a alors, pour chaque côté T de $\mathcal{C}_h$, l'existence de 2 triangles K et K' de $\mathcal{T}_h$ tels que :

$$K \neq K' \quad et \quad T = \partial K \cap \partial K' \tag{24}$$

A toute fonction $v \in L^2(\Omega)$, on associe $v\,|_K = v_K \in L^2(K)$ pour tout $K \in \mathcal{T}_h$.

Soient $K \in \mathcal{T}_h$ et $v_K \in H^2(K)$, on note $\partial_n^K v_K$ la dérivée normale de v_K sur ∂K.

Sous les notations (24), on définit alors le saut de la dérivée normale de v sur tout côté T de $\mathcal{T}_h$ par

$$[\partial_n v]_T = \begin{cases} \partial_n^K v_K + \partial_n^{K'} v_{K'} & \text{p.p. sur } T, \text{ si } T \in \mathcal{C}_h, \\ 0 \quad \text{sur } T, \text{ si } T \text{ est un côté frontalier.} \end{cases} \tag{25}$$

On remarque que si $v \in H^2(\Omega)$ alors $[\partial_n v]_T = 0, \quad \forall T \in \mathcal{C}_h$.

On considère les espaces discrets suivants :

$$\begin{aligned} Y_h &= \left\{\theta_h \in \mathcal{C}^0(\overline{\Omega}) ; \forall K \in \mathcal{T}_h , \theta \mid_K \in P_1(K)\right\}. \\ Y_h^0 &= Y_h \cap \mathrm{H}_0^1(\Omega). \end{aligned} \tag{26}$$

où $\mathcal{C}^0(\overline{\Omega})$ l'espace des fonctions continues sur $\overline{\Omega}$ et $P_1(K)$ est l'espace des fonctions polynômiales de degré inférieur ou égal à 1, définies dans $\overline{K}$.

Notons par ω_h, ψ_h et f_h respectivement les approximations de ω, ψ et f dans Y_h. La formulation variationnelle approchée du problème (21), par la méthode des éléments finis classique, consiste à trouver ψ_h et ω_h solutions de

$$\begin{cases} a(\omega_h, \theta_h) + b(\theta_h, \psi_h) = < g, \theta_h >_{-\frac{1}{2},\frac{1}{2},\Gamma} & \forall \theta_h \in Y_h \\ b(\omega_h, \eta_h) + d(\psi_h, \eta_h) = < f, \eta_h >_{-1,1,\Omega} & \forall \eta_h \in Y_h^o \\ \psi_h - \psi_d \in Y_h^o \ , \omega_h \in Y_h \end{cases} \tag{27}$$

où a, b et c sont trois formes bilinéaires définies par :

$$\begin{aligned} a(\delta_h, \theta_h) &= \int_{\Omega_h} \theta_h \, \delta_h \, \mathrm{d}\Omega_h \qquad \forall \quad \delta_h \in Y_h, \quad \theta_h \in Y_h \\ b(\theta_h, \eta_h) &= \int_{\Omega_h} \nabla\theta_h \, \nabla\eta_h \, \mathrm{d}\Omega_h \quad \forall \quad \theta_h \in Y_h, \quad \eta_h \in Y_h^0 \\ d(\phi_h, \eta_h) &= \int_{\Omega_h} \nabla\phi_h \, \nabla\eta_h \, \mathrm{d}\Omega_h \qquad \forall \phi_h \in Y_h, \quad \eta_h \in Y_h^0 \end{aligned} \tag{28}$$

Avec cette approche discrète, on remarque que la forme bilinéaire $a(.,.)$ n'est pas uniformément coercive; cela entraîne la perte d'un ordre d'erreur dans ce type de méthode. Pour pallier à cet aspect, on utilise la méthode $\mathcal{C}^0$ optimale.

3.2.2 Méthode $\mathcal{C}^0$ optimale

La méthode $\mathcal{C}^0$ optimale est une méthode d'éléments finis basée sur la technique de stabilisation décrite par Amara [4] et Amara-Dabaghi [5] pour le problème de Stokes, par Amara et al [3][6] pour le problème de Quasi-Stokes. Essentiellement, elle consiste à changer la forme principale $a(.,.)$ en une autre forme $a_h(.,.)$ en ajoutant un terme de stabilisation. La nouvelle forme $a_h(.,.)$ est alors uniformément coercive sur le problème discret et cette nouvelle formulation reste consistante. Dans cette étude, on se base sur ces travaux; en effet, on ajoute à la forme $a(.,.)$ un terme qui tient compte des sauts des dérivées normales sur les côtés internes de la triangulation. Ce terme est défini par la forme bilinéaire $\langle .\,,\, . \rangle_h$ suivante:

$$\langle \theta_h \,,\, \delta_h \rangle_h = \sum_{T \in \mathcal{C}_h} mesT \int_T [\partial_n \theta_h]\,[\partial_n \delta_h]\,\mathrm{d}T, \quad \forall \theta_h \in Y_h,\ \forall \delta_h \in Y_h \tag{29}$$

avec $mesT$ est la mesure du côté T de $\mathcal{C}_h$.

La nouvelle forme a_h s'écris alors;

$$\mathrm{a}_h(\delta_h, \theta_h) = \mathrm{a}(\delta_h, \theta_h) + \beta_h \,\langle \delta_h \,,\, \theta_h \rangle_h \,, \quad \forall \theta_h \in Y_h,\ \forall \delta_h \in Y_h \tag{30}$$

avec β_h est un paramètre positif de stabilisation.

La formulation discrète du problème s'écrit donc

$$a_h(\delta_h, \theta_h) = a(\delta_h, \theta_h) + \beta\, A_h$$

où

$$A_h(\delta_h, \theta_h) = \sum_{T \in \mathcal{C}_h} h_T \int_T [\partial_n \delta_h]\,[\partial_n \theta_h]\,\mathrm{d}T$$

$\beta \geq 0$ est un paramètre de stabilisation donné.

On obtient ainsi une nouvelle formulation du problème (21)

$$\begin{cases} a_h(\omega_h, \theta_h) + b(\theta_h, \psi_h) \;\; = < g, \theta_h >_{-\frac{1}{2},\frac{1}{2},\Gamma} & \forall \theta_h \in\ Y_h \\ b(\omega_h, \eta_h) + d(\psi_h, \eta_h) \;\; = < f, \eta_h >_{-1,1,\Omega} & \forall \eta_h \in Y_h^o \\ \psi_h - \psi_d \in Y_h^o \;\; , \omega_h \in Y_h \end{cases} \tag{31}$$

Le problème (31) admet une solution unique (Amara et *al.* [6]).

4 Système matriciel

Soit n le nombre de noeuds de la discrétisation, $(\theta_i)_{i=1,n}$ une base de Y_h et $(\eta_i)_{i=1,n}$ une base de Y_h^0. Le système (31) s'écrit pour chaque $j = 1, ..., n$

$$\begin{cases} \sum_{i=1}^{n} \omega_i \, a(\theta_i, \theta_j) + \beta \sum_{i=1}^{n} \omega_i \, A_h(\theta_i, \theta_j) + \sum_{i=1}^{n} \psi_i \, b(\theta_j, \eta_i) =< g, \theta_j > \\ \sum_{i=1}^{n} \omega_i \, b(\theta_i, \eta_j) + \sum_{i=1}^{n} \psi_i \, d(\eta_i, \eta_j) =< f, \eta_j > \end{cases} \tag{32}$$

qui s'écrit sous la forme matricielle suivante :

$$\begin{cases} (A \ + \beta S)\, \omega \quad + \quad B^T \ \psi = \ G \\ -B \ \omega \qquad\quad + \quad D \ \psi = \ F \end{cases} \tag{33}$$

avec

- A est la matrice de masse associée à la forme bilinéaire $a(.,.)$,
- S est la matrice de saut associée à A_h,
- B est la matrice de rigidité associée à la forme bilinéaire $b(.,.)$,
- $D = -\lambda B$,
- G et F représentent les seconds membres.

ce qui nous donne un système linéaire à résoudre de la forme

$$MX = N \tag{34}$$

avec

$$M = \begin{pmatrix} A + \beta S & B^T \\ -B & -\lambda B \end{pmatrix} \quad , \quad X = \begin{pmatrix} \omega \\ \psi \end{pmatrix} \quad \text{et} \quad N = \begin{pmatrix} G \\ F \end{pmatrix}$$

où $A + \beta S$ et B sont symétriques et définies positives.

Pour le stockage de ces matrices, la méthode Morse est utilisée. Il consiste à introduire deux pointeurs ligne et colonne qui permettent de retrouver la valeur d'une composante de la matrice à partir d'un tableau où seulement les éléments non nuls sont stockés.

5 Le solveur scalaire

Dans ce paragraphe, on présente l'algorithme général du code qui se ramène à la résolution du système (34)

5.1 Algorithme

L'organigramme de calcul du code est donné dans la figure 2. Les étapes de l'algorithme sont les suivants :

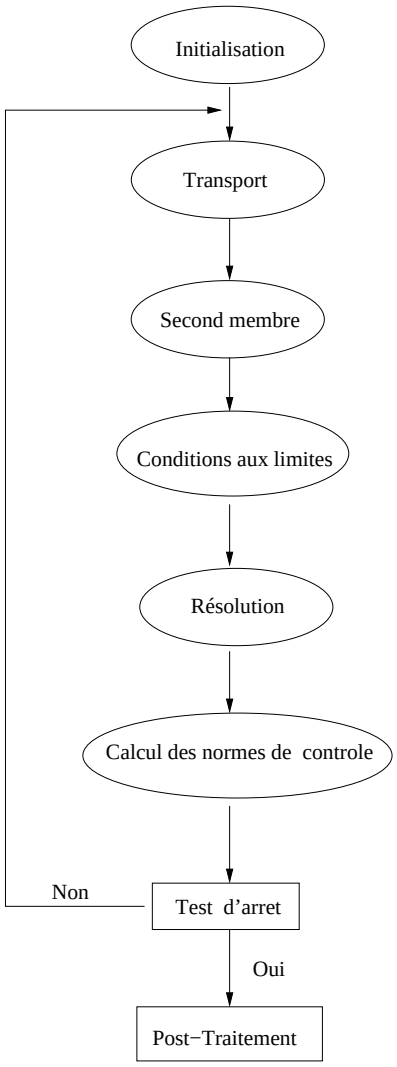

FIG. 2 – *Algorithme $\psi - \omega$*

5.1.1 Initialisation

On commence par la lecture des données utilisées pour le calcul (maillage, paramètres liés au problème traité). Ensuite, on procède à la construction des tables géométriques (pointeurs ligne et colonne utilisés pour le stockage, éléments voisins des noeuds, éléments voisins des éléments, etc). Enfin, on calcule les matrices du système à résoudre (masse, rigidité, saut).

5.1.2 Transport

Transport de ω par la méthode des caractéristiques dont l'algorithme de résolution est présenté en détail dans [1].

5.1.3 Second membre

Calcul du second membre qui dépend des grandeurs transportées.

5.1.4 Conditions aux limites

Imposition des conditions aux limites sur la matrice et le second membre.

5.1.5 Résolution

M est une matrice carrée d'ordre $2n$, définie positive mais non symétrique. La méthode choisie pour la résolution du système linéaire (34) est celle du double gradient conjugué (BICG). Elle consiste à appliquer un algorithme de gradient conjugué sur le système suivant:

$$\mathcal{M}\mathcal{X} = \mathcal{N}$$

où $\mathcal{M} = \begin{bmatrix} 0 & M^T \\ M & 0 \end{bmatrix}$, $\mathcal{X} = \begin{Bmatrix} X \\ X^\star \end{Bmatrix}$, $\mathcal{N} = \begin{Bmatrix} N \\ N^\star \end{Bmatrix}$.

Ici $X^\star$ représente une inconnue fictive et $N^\star$ est un second membre fictif. En tenant compte de ce changement de variable, la matrice ainsi obtenue est symétrique, définie positive.

La rapidité de convergence d'une méthode itérative dépend énormement du nombre de conditionnement de la matrice (le conditionnement d'une matrice M est défini par $K(M) = \|M\|\|M^{-1}\|$). Plus $K(M)$ est proche de 1, plus vite converge l'algorithme.
Le principe du préconditionnement d'une matrice consiste à remplacer la résolution de l'équation $MX = N$ par celle du système équivalent

$$C^{-1}MX = C^{-1}N \tag{35}$$

et que C^{-1} soit choisie avec l'objectif que $K(C^{-1}M)$ soit beaucoup plus petit que $K(M)$. En théorie, le meilleur choix est évidement que $C^{-1} = M^{-1}$ ce qui donne $K(C^{-1}M) = 1$.

En pratique, il faudra trouver C^{-1} le plus proche de M^{-1}, sans que les calculs de C^{-1} soient trop coûteux.
La recherche d'un préconditionneur adéquat nécessite toute une étude à part entière. Cependant, pour la validation de notre algorithme on a utilisé la diagonale de la matrice.

L'algoritme BICG préconditionné appliqué au système (35) est donné par :

- Initialisation
  - $r_0 = N - MX_0$
  - choisir r_0^* tel que $(r_0, r_0^*) \neq 0$
  - $P_0 = C^{-1} r_0$, $P_0 = C^{-1} r_0^*$
- Pour $k = 1, 2...$ jusqu'à convergence
  - $\alpha_k = (C^{-1} r_k, r_k^*)/(MP_k, P_k^*)$
  - $X_{k+1} = X_k + \alpha_k P_k$
  - $r_{k+1} = r_k - \alpha_k M P_k$
  - $r_{k+1}^* = r_k^* - \alpha_k M^T P_k^*$
  - $\beta_k = (C^{-1} r_{k+1}, r_{k+1}^*)/(C^{-1} r_k, r_k^*)$
  - $P_{k+1} = C^{-1} r_{k+1} + \beta_k P_k$
  - $P_{k+1}^* = C^{-1} r_{k+1}^* + \beta_k P_k^*$

5.1.6 Calcul des normes de contrôle

On calcule les normes qui sont utilisés pour tester la convergence du code.

5.1.7 Test d'arrêt

Le programme s'arrête dès que la convergence ou le nombre maximal d'itérations autorisé soient atteints.

5.1.8 Post-traitement

Stockage des solutions calculées dans des fichiers pour d'éventuelles applications graphiques.

5.2 Coûts informatiques

La simulation numérique du processus d'aération sur un cas d'application réelle nécessite des ressources très importantes en temps calcul et en place mémoire. Prenons par exemple le cas d'une simulation dans une coupe d'un vrai lac (voir [1]) dont les dimensions moyennes sont de l'ordre de $500\,m$ x $20\,m$. L'injecteur est de l'ordre de $10\,m$ et comporte une centaine de trous de l'ordre de $1\,cm$ chacune. Si on veut détecter convevablement l'effet de l'injection des bulles, il est nécessaire de faire un maillage avec des éléments de l'ordre du volume de la bulle ce qui donnera un maillage d'une dizaine de millions de noeuds. Si on propose de

faire un maillage fin sur une zone proche de l'injecteur et grossier ailleurs, on reste toujours dans l'ordre de quelques millions de noeuds. Pour fixer les idées, un calcul sur un maillage de $6\,10^5$ noeuds nécessite environ 3h30mn pour une itération en temps sur une machine HP 240Mhz et 490 Mb de mémoire. Pour 1 heure de temps de simulation avec un pas de temps d'une minute, il faut 60 itérations en temps soit 210 heures de calcul ($\simeq$ 9 jours). Pour des simulations plus coûteuses, la parallélisation du code est incontournable.

6 Parallélisation

L'implémentation parallèle du code a deux objectifs :

- Obtenir des temps de calculs raisonnables. Nous pouvons espérer idéalement diviser le temps réel (temps Elapsed) par le nombre de processeurs utilisés.
- Pouvoir traiter des simulations réelles exigeant une très importante place mémoire, en répartissant le stockage des données dans la mémoire de chaque processeur.

Après une présentation de la machine utilisée pour faire les simulations, on présente en détail l'algorithme parallèle. Une analyse des performances est ensuite donnée.

6.1 Présentation de la HP 9000 V-Class

La HP 9000/800 à été une des plate formes les plus rapides fournies par Hewlett-Packard (HP) en l'an 2000. La machine, également appelée serveur V-Class, est une machine M.I.M.D (multiples instructions, multiples données) et possède une architecture de mémoire partagé [17].

6.1.1 Configuration interne de la machine

La V-Class contient 16 processeurs. Les processeurs et le matériel associés forment ce qui s'appelle généralement le noeud. Le noeud utilise une conception symétrique du multi-processeur (SMP) qui peut exploiter le parallélisme à grain fin. Un schéma fonctionnel conceptuel de l'architecture interne de la machine est donné dans la figure 3. Dans ce schéma, on voit que la machine est synchronisée par plusieurs unités de comande.

- Centralement située dans le diagramme, la barre transversale d'hyperplan de HP qui est composée de quatre contrôleurs de connexion de routage (ERAC). Elle permet à tous les processeurs d'accéder à toute la mémoire disponible.
- Les processeurs sont installés sur les agents contrôleurs des processeurs (EPAC). Un EPAC permet au processeur et au sous-ensemble d'Entrée/Sortie (le contrôleur d'interface EPIC) l'accès à la barre transversale d'hyperplan. Jusqu'à deux processeurs sont situés sur chaque EPAC.
- Également reliés à la barre transversale d'hyperplan sont les contrôleurs d'accés mémoire (EMAC).
- Les entrées/sorties se relient au système par l'EPIC qui est reliée aux agents de processeur (EPAC).

– L'utilitaire de noyau dans le noeud (ECUB, généralement appelé panneau d'utilitaires) contient une section du matériel (hardware) appelée logique de noyau. Elle fournit des interruptions à tous les processeurs du système par le bus de logique de noyau qui se relie à chaque agent de processeur.

Le processeur et la mémoire sont reliés par un commutateur à barres croisées utilisant quatre ERAC de sorte que chaque processeur soit relié à toute la mémoire. Chaque ERAC a une largeur de bande de 1,9 GB/s (en gigaoctets par seconde), de sorte que la largeur de bande globale de l'ensemble des quatre ERAC soit 15 GB/s. La figure 4 montre un diagramme du commutateur à barres croisées.

6.1.2 Unités de traitement

La V-Class utilise les processeurs PA-8200 de HP. Chaque processeur est basé sur l'architecture RISC (Reduced Instruction Set Computer) à 64 bits de fréquence 240 MHz. Il est doté d'une puissance en crête de 960 MegaFlops (MF: million d'opérations de virgule flottante par seconde).

6.1.3 Organisation de la mémoire

Mémoire Physique: La mémoire physique ou la RAM (mémoire à accès sélectif) sur la V-Class est de 8GB de stockage (extensible à 16GB) répartis sur 64 banques de mémoire de 64 Mb. La division de la mémoire (RAM) dans 64 banques de mémoire permet l'accès simultané de la mémoire vers les différentes banques, ce qui ne serait pas possible si la mémoire entière était traitée en tant qu'une unité de mémoire. Chaque EMAC commande un ensemble de huit banques de mémoire avec un total de 512 Mb de mémoire. La mémoire physique a une latence de 500 nanosecondes, approximativement. La mémoire physique contient les programmes à exécuter et les données (appelés "mémoire logique"), mais contient également beaucoup d'informations “système”.

Mémoire virtuelle: La mémoire virtuelle est disponible par l'utilisation d'unités de mémoire appelées pages. Elle permet, par l'accès à des segments de page de mémoire, l'exécution de beaucoup plus de programmes que ne peut la mémoire physique. Le compilateur produit des adresses de la mémoire virtuelle beaucoup plus grandes, mais seules les pages s'exécutant sont traduites en mémoire physique. La HP PA-RISC 8200 traduit des adresses de mémoire virtuelles et physiques de 32 bits et 64 bits en adresses de 64 bits.

Cache: Chaque processeur possède 2 Mb de mémoire cache pour les données ainsi que 2 Mb de mémoire cache pour les instructions, soit un total de 4 Mb. La mémoire cache est certes petite mais plus rapide (10 fois plus) que la mémoire physique pour l'accès à l'unité centrale de traitement.

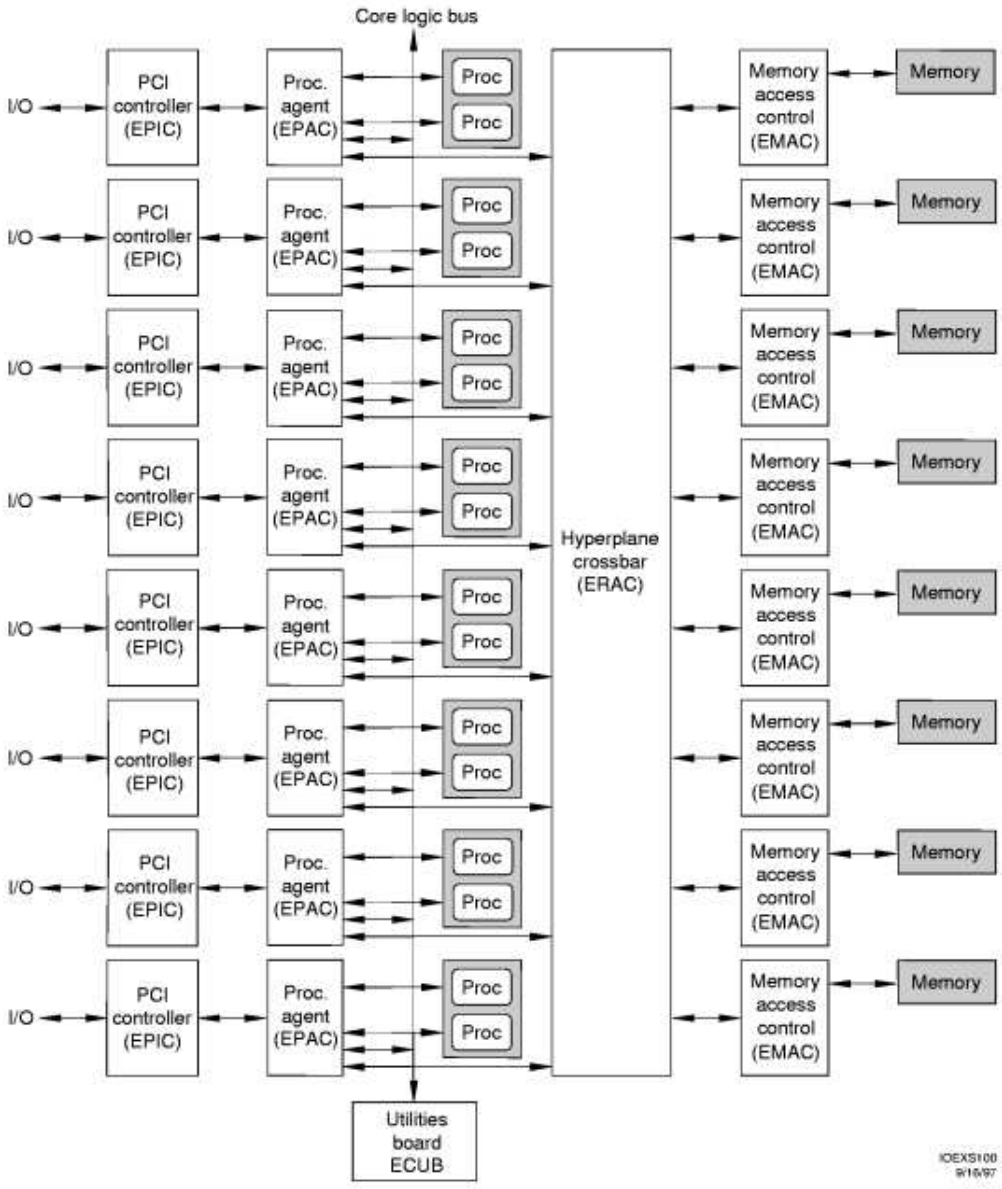

FIG. 3 – *Architecture interne de la V-Class*

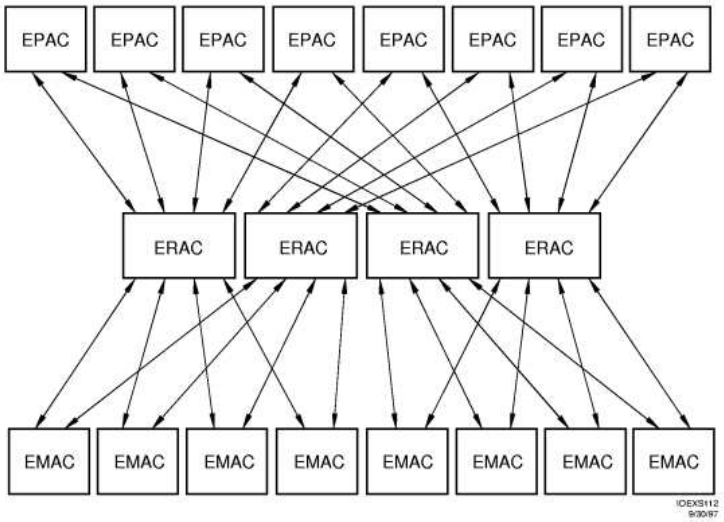

FIG. 4 – *Diagramme de la configuration de l'ERAC*

6.1.4 Programmation parallèle

Le système d'exploitation de la V-Class est le HP-UX 11.0. Sur le plan des logiciels, la plupart des langages de programmation sont accessibles sur cette machine.

Pour la programmation parallèle, le HP MPI (Message Passing Interface) est utilisé afin de rendre le solveur portable sur n'importe quelle machine parallèle virtuelle et donc non nécessairement à mémoire partagée. MPI fournit un large éventail de dispositifs facilitant le développement des applications parallèles parmi lesquels :

- La conformité à la norme de la version 1.2 de MPI.
- Des styles de programmation SPMD (programme simple à multiples données) et MPMD (programmes multiple à multiples données).
- Un outil (XMPI) permettant de tracer et de surveiller les applications et le fonctionnement des processeurs.
- Un outil (MPIVIEW) permettant de visualiser et d'analyser les performances de chaque processeur.

6.2 Analyse algorithmique et implémentation

Des tests numériques effectués ont montré que la partie résolution du système linéaire consomme environ 90% du temps CPU total du code sur un ordinateur mono-processeur. Par conséquent, on s'est focalisé uniquement sur le traitement algorithmique parallèle de cette partie du code qui utilise le solveur itératif BICG avec préconditionneur diagonal.

L'application de la méthode BICG nécessite un grand nombre de multiplications matrice-vecteur. Cette opération est de loin la plus importante au niveau du temps de résolution du problème; d'où l'idée de partager cette opération entre les processeurs afin de minimiser au maximum le temps de calcul.

6.2.1 BICGSTAB au lieu du BICG

L'handicap majeur dans la parallélisation de la méthode BICG est l'utilisation de M^T, le transposé de M. En effet, l'utilisation de M^T nécessite soit son stockage et par conséquent doubler la mémoire, soit faire appel à M. La difficulté réside dans le fait que M sera partagée sur les processeurs et la restitution de M^T engendrera des temps de communication entre processeurs extrêmement importants; ceci aura pour effet d'augmenter le temps système et par conséquent pénaliser la performance. Pour palier à cet handicap, l'alternative consiste à utiliser une autre méthode équivalente, celle du double gradient conjugué stabilisé (BICG-STAB). En effet, cette variante ne nécessite plus d'une part que la matrice M et d'autre part que deux multiplications Matrice-Vecteur au lieu de trois. Son algorithme est donné par :

- Initialisation
  - $r_0 := C^{-1}(N - MX_0)$
  - choisir r_0^* tel que $(r_0, r_0^*) \neq 0$
  - $P_0 = V_0 = 0$
  - $\eta_0 = \alpha_0 = w_0 = 1$
- Pour k=1, 2, ... jusqu'à convergence
  - $\eta_{k+1} = (r_k, r_0^*)$
  - $\beta_{k+1} = \alpha_k \rho_{k+1} / \rho_{k+1} w_k$
  - $P_{k+1} = r_k + \beta_{k+1}(P_k - w_k V_k)$
  - $V_{k+1} = C^{-1} M P_{k+1}$
  - $\alpha_{k+1} = \rho_{k+1} / (r_0^*, V_{k+1})$
  - $S_{k+1} = r_k - \alpha_{k+1} V_{k+1}$
  - $T_{k+1} = C^{-1} M S_{k+1}$
  - $w_{k+1} = (T_{k+1}, S_{k+1}) / (T_{k+1}, T_{k+1})$
  - $X_{k+1} = X_k + \alpha_{k+1} P_{k+1} + w_{k+1} S_{k+1}$
  - $r_{k+1} = S_{k+1} - w_{k+1} T_{k+1}$

6.2.2 Dérivation des tâches indépendantes :

On rappelle qu'une tâche est considérée indépendante si le résultat de son travail est non lié à celui d'une autre.

Soit A une matrice symétrique d'ordre n stockée en mode Morse, X un vecteur donné, AX le produit de la matrice A par le vecteur X et IT, JT représentant respectivement les

pointeurs ligne et colonne du stockage Morse. Étant donnée la manière avec laquelle sont stockées les matrices, le produit matrice vecteur en mode séquentiel se fait comme suit:

- Faire $I = 1, n$
  - Faire $K = IT(I-1)+1, IT(I)-1$
    - $AX(I) = AX(I) + A(K).X(JT(K))$
    - $AX(JT(K)) = AX(JT(K)) + A(K).X(I)$
  - Fin faire
  - $AX(I) = AX(I) + A(IT(I)).X(I)$
- Fin faire

Ce schéma de calcul semble incompatible avec l'objectif de faire le calcul en parallèle à travers des tâches indépendantes. La solution choisie consiste à changer la manière de stocker les données afin de rendre le partage des tâches possible tout en gardant la cohérence du résultat. Pour réaliser cet objectif, deux choix essentiels ont été faits :

- Ne pas tenir compte de la symétrie des matrices,
- Duplication du vecteur X sur tous les processeurs.

Ces deux choix, certes coûteux au niveau de la place mémoire mais de manière raisonable, assurent un minimum d'indépendance entre les différentes tâches.

6.2.3 Opérations parallélisées

Dans l'algorithme BICGSTAB, on effectue pour chaque itération :

- 2 produits matrice-vecteur,
- 4 produits scalaires,
- 6 combinaisons linéaires.

Ces opérations sont effectués comme suit :

Produit matrice-vecteur: La stratégie utilisée est d'équirépartir les lignes de la matrice entre les processeurs. Cette partition, paramétrée en fonction du nombre de processeurs dès le lancement de l'exécution, est optimale dans le cas d'une machine parallèle virtuelle constituée d'un parc de processeurs homogènes. Ainsi, chaque processeur contient le vecteur global et un certain nombre de lignes de la matrice comme le montre la figure 5.

Chaque processeur P_i effectue localement le produit d'un bloc A_i (composé d'un certain nombre de lignes de la matrice A) et d'un vecteur X. Le résultat est un vecteur $(AX)_i$ de dimension le nombre de lignes du bloc A_i. Par conséquent, aucune communication n'est nécessaire dans le calcul local du produit matrice-vecteur. En fait, cette répartition met en œuvre des messages de communication courts; en effet à chaque itération elle charge le vecteur X (dupliqué sur tous les processeurs) et communique à la fin le vecteur $(AX)_i$ vers les autres processeurs.

Combinaison linéaire de deux vecteurs: Chaque processeur contient une partie de chaque vecteur global impliqué dans l'opération (P_{k+1}, S_{k+1}), dit vecteur local. Le vecteur résultat

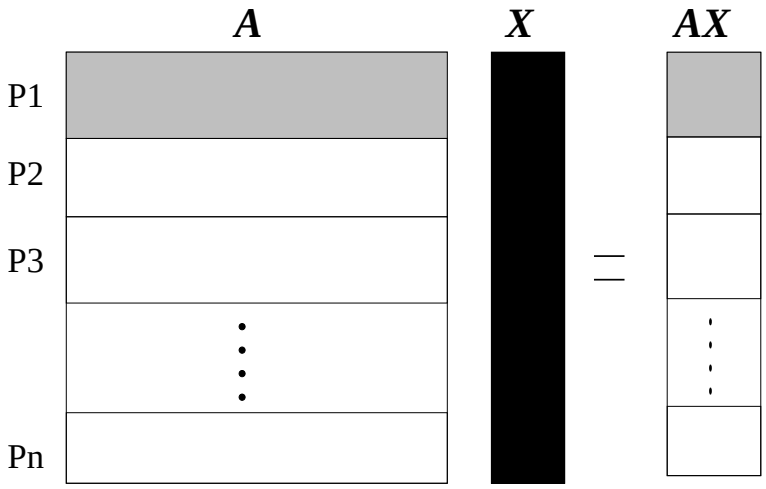

FIG. 5 – *Produit matrice vecteur*

de la combinaison linéaire obtenu est aussi un vecteur local. Par conséquent, aucune communication n'est nécessaire dans cette opération.

Produit scalaire: Les quatre produits scalaires de cet algorithme représentent des données globales mettant en jeu des composantes réparties de vecteurs locaux. Ainsi, chaque processeur calcule un produit scalaire partiel de deux vecteurs locaux correspondants en multipliant ces deux vecteurs et en sommant les composantes du résultat partiel. Ensuite chaque processeur envoie à tous les autres son résultat partiel et reçoit les produits des autres processeurs qu'il somme à son propre résultat. Par conséquent, une communication globale est nécessaire dans ce produit scalaire. *MPI* propose une routine () pour opérer la sommation sur les données réparties sur l'ensemble de processeurs avec récupération du résultat sur tous.

6.2.4 Contiguïté des données et overhead

La plus grande partie d'overhead résulte généralement des échanges (envoi et réception) des messages entre tâches lors de la phase de réception/diffusion des données. Pour cela, on commence par la distribution de toutes les données nécessaires au calcul; ceci dit, avant de commencer le calcul, une tâche recevra un paquet de données contenant la partie utile de A, IT, JT et le vecteur X tout entier. Tous les échanges effectués après cela ne concernent que la mise à jour des vecteurs de directions P_k et S_k dans l'algorithme BICGSTAB. Des communications synchrones sont utilisées dans la mise à jour des vecteurs globaux redistribués après chaque produit matrice-vecteur, via la routine de collecte générale : *MPI_ ALLGATHER*.

6.3 Algorithme général

L'organigramme du code parallèle est donné dans la figure 6. L'algorithme parallèle de résolution du système linéaire est donné ci-après.

6.3.1 Intialisation

On commence par intialiser l'environnement de calcul MPI.

6.3.2 Préparation

Le processeur maître prépare le système linéaire à résoudre. Ceci correspond aux étapes transport, second membre et conditions aux limites dans l'algorithme séquentiel.

6.3.3 Partage & envoi

Le processeur maître prépare et envoie à chaque processeur esclave et à lui même les parties de données nécessaires pour effectuer ses différentes tâches de calcul.

6.3.4 Résolution

- Initialisation
  - Calcul local de η_0
- Pour k=1, 2, ... jusqu'à convergence
  - Calcul local de η_{k+1} puis sommation globale (*MPI_ALLREDUCE*) avec les autres processeurs
  - Calcul de β_{k+1}
  - Calcul local de P_{k+1}
  - Mise à jour (*MPI_ALLGATHER*) de P_{k+1} dans tous les processeurs
  - Calcul local de V_{k+1}
  - Calcul local du produit scalaire (r_0^*, V_{k+1}) puis sommation globale (*MPI_ALLREDUCE*) avec les autres processeurs
  - Calcul de α_{k+1}
  - Calcul local de S_{k+1}
  - Mise à jour (*MPI_ALLGATHER*) de S_{k+1} dans tous les processeurs
  - Calcul local de T_{k+1}
  - Calcul local du produit scalaire (T_{k+1}, S_{k+1}) puis sommation globale (*MPI_ALLREDUCE*) avec les autres processeurs
  - Calcul local du produit scalaire (T_{k+1}, T_{k+1}) puis sommation globale (*MPI_ALLGATHER*) avec les autres processeurs
  - Calcul de w_{k+1}
  - Calcul local de X_{k+1}
  - calcul local de r_{k+1}

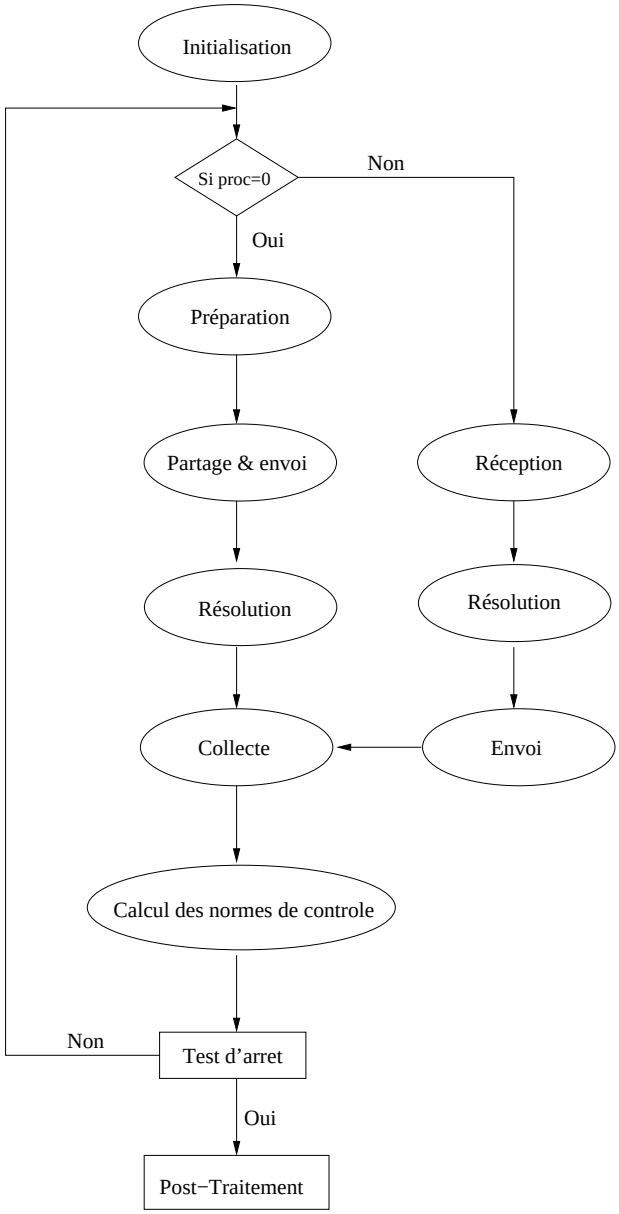

FIG. 6 – *Algorithme parallèle*

6.3.5 Réception

Les processeurs esclaves reçoivent du processeur maître leurs parts de données nécessaires pour effectuer localement leurs tâches.

6.3.6 Résolution

Les processeurs esclaves effectuent chacun sa part de calcul dans l'algorithme BICG-STAB. L'algorithme est le même que celui effectué par le processeur maître.

6.3.7 Envoi

Les processeurs esclaves envoient leurs résultats finaux et locaux au processeur maître.

6.3.8 Collecte

Le processeur maître fait la collecte du résultat final. Ceci correspond à la récupération de toutes les parties de X_{k+1} calculées par chaque processeur dans l'algorithme BICGSTAB.

6.4 Présentation du cas test et Performances

6.4.1 Cas test

On s'intéresse au comportement du code de calcul en fonction de la dimension des problèmes traités. Pour cela, on considère un domaine carré $[0,1] \times [0,1]$ discrétisé suivant les trois maillages décrits dans le tableau ci-dessous :

Maillage	nombre de Noeuds	nombre d'éléments	nombre d'inconnues
h_1	48x48	4418	4608
h_2	96x96	18050	18432
h_4	192x192	72962	73728

TAB. 1 – *Maillages cavité*

On note, qu'en passant du cas h_1 au cas h_2, ou du cas h_2 au cas h_4, on traite quatre fois plus de données.

6.4.2 Outils de performance

Les résultats des tests seront analysés à travers les deux types de courbes suivants :

- les courbes de temps qui nous informent sur l'évolution du temps d'exécution du programme en fonction du nombre de processeurs ainsi que sur l'importance du coût dû à la communication,
- les courbes d'accélération (*Speed-Up*).

En fait, c'est à travers les courbes de *Speed-Up* qu'on pourra apprécier principalement l'efficacité d'un programme parallèle. Si le programme séquentiel résout un problème de taille n en un temps $T_1(n)$ et si l'algorithme parallèle le résout en un temps $T_p(n)$ avec p processeurs, l'accélération (*Speed-Up*) est donnée par le nombre suivant :

$$S_p(n) = \frac{T_1(n)}{T_p(n)}$$

Ce terme peut être inférieur à 1 si le programme parallèle est pénalisé par le surcoût lié à la communication entre les processeurs dû à l'utilisation des routines de communication. On distingue deux parties dans le temps de communication

- *l'overhead stable* qui représente le coût des send/receive,
- *l'overhead instable* qui mesure le temps des attentes réseau. Celui-ci est important lors de l'utilisation d'une machine parallèle virtuelle (réseau de stations sous intranet ou internet) mais reste incontrôlable puisqu'il est intimement lié à l'état du réseau (charge, bande passante,...).

Par la suite, on considère le temps *Elapsed* qui est la somme du temps de calcul et celui de communication (sans oublier qu'il dépend étroitement de la charge de la machine). Le temps de communication ou *overhead* comprend le temps de préparation (latence + surcoût) et le temps de transfert.

6.4.3 Analyse des performances

On présente dans la figure 7 le temps de simulation des trois maillages en fonction du nombre de processeurs. On remarque que plus le maillage est fin, i.e. le nombre d'inconnues est important, plus le gain en temps devient important.

Ce gain en temps est proportionnel au cas traité et est souvent pollué par le temps de communication entre les processeurs qui devient plus important quant on augmente le nombre de processeurs. Dans la figure 8, on montre les courbes de speed-up pour les trois maillages. Dans les trois cas, on remarque qu'à partir d'un certain nombre de processeurs, l'efficacité décroît à cause du temps de communication qui devient important. Néanmoins, on observe aussi que cette décroissance est moindre lorsque le nombre d'inconnus à traiter est plus important. Ceci montre l'influence de la granularité des données sur les temps de calcul. On note aussi que la tendance à un comportement super linéaire sur les courbes du Elapsed speed-up est dûe essentiellement au cache de la machine.

Dans la suite, on va se pencher de plus près sur l'analyse du cas h_4. La figure 9 montre les résultats obtenus au niveau du temps de calcul et de l'overhead. On peut voir l'influence de celui-ci en regardant les courbes de speed-up (Fig 10). En effet, l'impact de l'overhead devient de plus en plus important lorsqu'on augmente le nombre de processeurs. On observe aussi une accentuation de l'effet dû au cache de la machine sur le Speed-up CPU ne tenant pas compte du temps système. De ce fait, et dans l'état actuel, il semble inutile d'aller au delà d'une dizaine de processeurs pour ce cas de calcul.

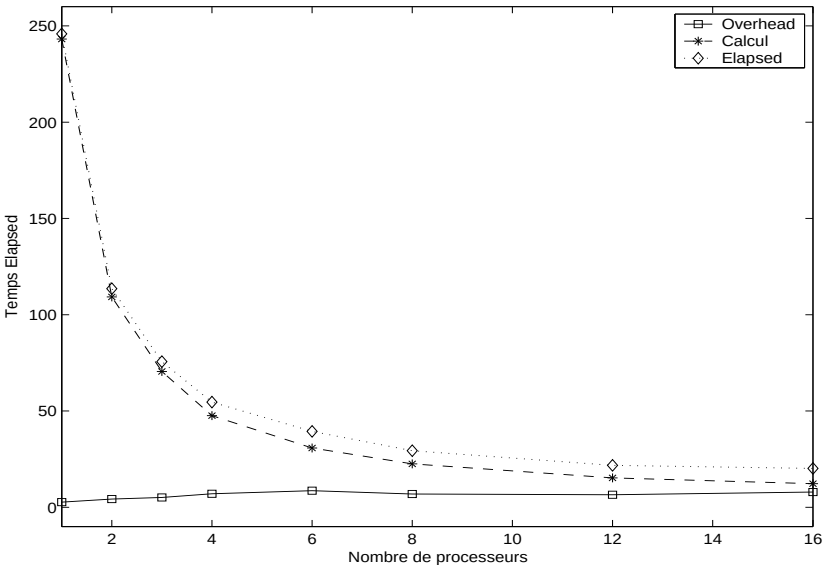

FIG. 7 – *Temps Elapsed sur la V-class*

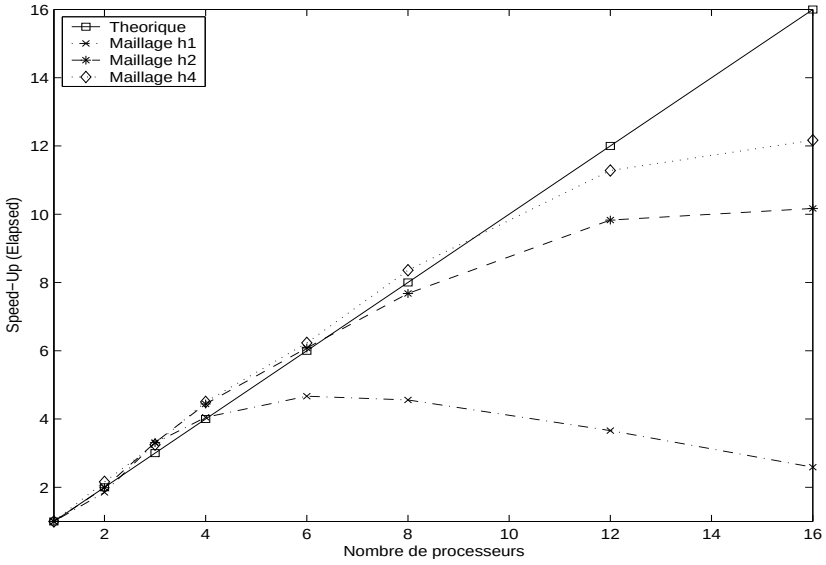

FIG. 8 – *Speed-Up sur la V-class*

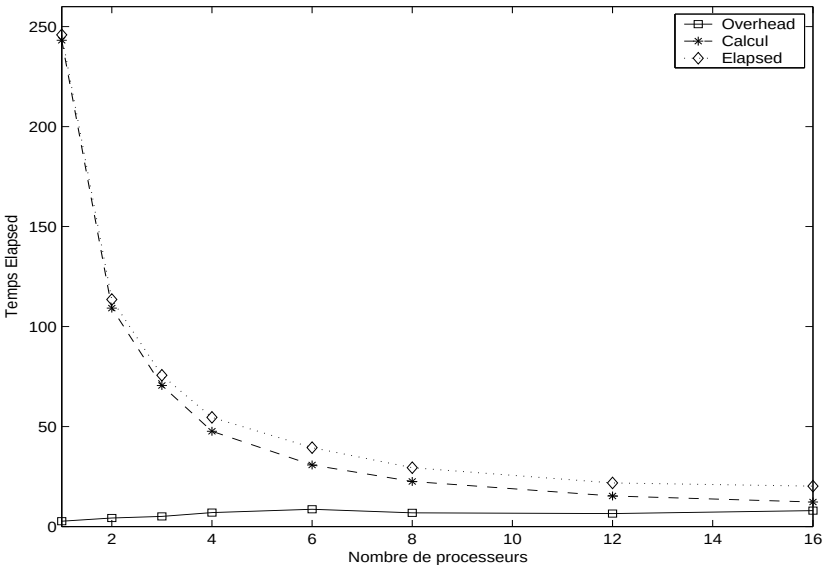

FIG. 9 – *Influence de l'overhead sur le temps de calcul*

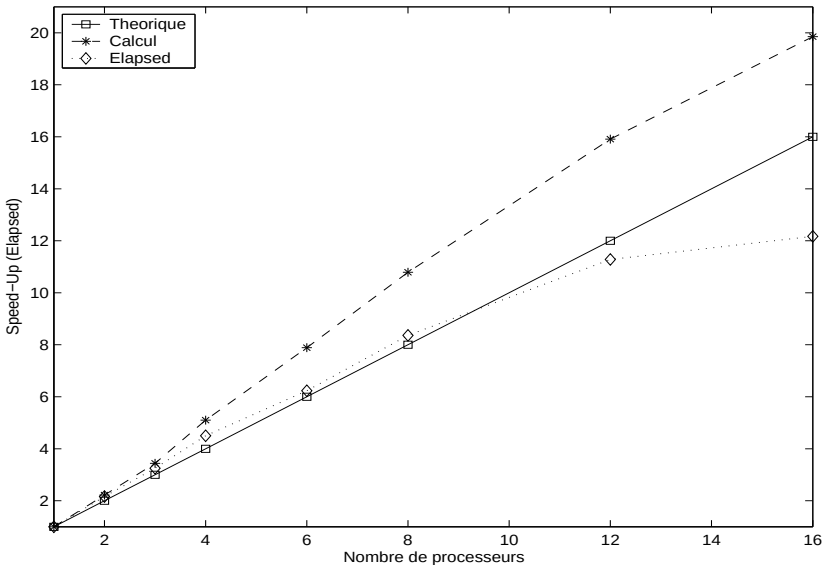

FIG. 10 – *Comparaison des speed-up calcul et elapsed*

7 Conclusion

Le but de ce travail était de répondre aux problèmes de place mémoire et de temps de calcul posés par la simulation du processus d'aération. Pour cette raison, on a présenté une première tentative de parallélisation d'un code numérique simulant ce problème par un modèle simplifié écrit en formulation courant-torbillon. Pour l'étude numérique, on a utilisé la méthode des caractéristiques pour la discrétisation spatiale et la méthode des éléments finis C^o optimal pour l'approximation spatiale. La parallélisation a ciblé le solveur BICGSTAB qui représente la partie la plus importante en temps de calcul. Cette stratégie adopté s'est révélée payante. En effet, les tests numériques effectués sur un cas d'application classique montrent des résultats de plus en plus convenables en terme de speed-up en augmentant la taille du problème traité. Du point de vue temps de calcul, on observe des gains très importants du code parallèle; ces gains étant fonction du nombre de processeurs utilisés. Ceci est très encourageant puisqu'il nous permet d'une part de traiter des cas d'application réels et d'autre part de généraliser cette alternative à des modèles plus complets tout en pensant à la parallélisation de la méthode des caractéristiques qui présente certes beaucoup de difficultés mais permettra d'avoir de meilleurs résulats. Notons au passage que ce travail mettait l'accent sur l'utilisation d'une machine parallèle dédiée sans pour autant perdre de vue l'aspect non dédié. En effet, sur le plan de la stratégie de répartition des lignes de la matrice, l'effort devra se focaliser sur une paramétrisation pondérée tenant compte d'une machine virtuelle parallèle hétérogène ou plus généralement d'une machine type Grid.

Remerciements

Ce travail a bénéficié du support de l'action intégrée franco-marocaine MA/01/03, du CMIFM et du projet communautaire ESIMEAU dans le cadre du $4^{ème}$ PCRD (ESPRIT-INCO). Les auteurs remercient le professeur D. Ouazar de l'Ecole Mohammadia d'Ingénieurs (Rabat-Maroc) pour son aide dans l'analyse du modèle de simulation numérique.

Références

[1] M. Abdelwahed, *Modélisation et simulation numérique d'écoulements diphasiques*, Thèse de doctorat, Université de Pau et Ecole Nationale d'Ingénieurs de Tunis, Octobre 2002.

[2] M. Abdelwahed, F. Dabaghi, D. Ouazar, *A virtual numerical simulation for aeration effects in lake eutrophication*, International journal on computational fluid dynamics, Vol. 16(2), p. 119-128, 2002.

[3] M. Abdelwahed, M. Amara, F. Dabaghi, M. Hassine, *A Numerical Modeling of a two phase flow for Water Eutrophication Problems*, Proceeding of the European Conference on Computational Methods in Applied Sciences, Eccomas 2000, 18 pages, Spain, 2000.

[4] M. Amara, *Une méthode optimale de classe C^0 d'approximation du bilaplacien*; Comptes Rendus de l'Académie des Sciences, Tome 319, p. 1327-1330, Série I, Paris 1994.

[5] Amara M., F. Dabaghi , *An optimal C° finite element method for the 2D biharmonic problem*, Numerisch Mathematik, 90, N°1, p. 19-46, 2001.

[6] M. Amara, F. Dabaghi, M. Hassine. *Une formulation mixte utilisant un élément fini C^0 optimal pour la résolution des équations de Quasi-Stokes incompressible: Analyse théorique et validation numérique*, RR INRIA 4142, Mars 2001.

[7] F. Brezzi, M. Fortin, *Mixed and hybrid finite element method*, Springer Series in Computational Mathematics, 15, Springer Verlag New York, 1991.

[8] F. Dabaghi. *Numerical Aspects of Aeration Process Modelling in Eutrophised Water Basins*, Journal of Systems Analysis Modelling Simulation, SAMS, Vol.39, p. 1-23, 2000.

[9] J. Douglas, T.F. Russel. *Numerical methods for convection dominated diffusion problems based on combining the method of characteristics with finite element methods or finite difference method*, SIAM, J. Numer. Anal. 19, p. 871-885, 1982.

[10] D.A. Drew, R.T. Lahey. *The three-dimensional time and volume averaged conservation equations of two-phase flow*, Advances in Nuclear sciences and Technology, Vol. 20, 1-69, 1988.

[11] H. Enwald, E. Peirano, A.E. Almstedt, *Eulerian two-phase flow theory applied to fluidization*, J. Multiphase flow Vol. 22, p. 21-66, 1996.

[12] V. Girault, P.A. Raviart, *Finite element methods for Navier-Stokes equations, Theory and Algorithms*, Springer Verlag Berlin, 1986.

[13] M. Ishii, , *Thermo-fluid dynamic theory of two-phase flow*, Collection de la direction des études de recherche d'électricité de france, EYROLLES 1975.

[14] O. T. Lind, *Reservoir eutrophication*, Selected proceedings of the 2nd International Conference on Reservoir Limnology and water quality, 9-14 August 1993, Ceské Budejovice, Czech Republic.

[15] O. Pironneau, *On the transport-diffusion algorithm and its application to the Navier-Stokes equations*, Numer. Math 38, p.309-332, 1982.

[16] E. Süli. *Convergence and non linear stability of the Lagrange-Galerkin Method for the Navier-Stokes Equations.* Numerishe Mathematik 53, p. 459-483, 1988.

[17] *The V-class server architecture*, http://docs.hp.com:80

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399